并行数据操作算法和查询优化技术*

李建中

(黑龙江大学信息技术研究所,哈尔滨 150080)

摘要 本文是并行数据库的查询处理并行化技术和物理设计方法”一文的续篇,继续综述并行数据库系统的另外两个重要研究领域:并行数据操作算法和并行数据库查询优化技术.最后,作为并行数据库系统研究与进展情况综述的结尾,本文将探讨并行数据库系统今后的研究方向和问题.

关键词 并行数据库,并行数据操作算法,并行数据库查询优化.

1 并行数据操作算法

许多研究表明,使用并行数据操作算法实现查询的并行处理可以更充分地发挥多处理机的并行性,更大地提高系统的效率和处理能力.并行数据操作算法是近几年最活跃的并行数据库系统研究领域.由于JOIN是关系数据库系统中最耗时且最常用的操作,并行数据操作算法的研究一直围绕着JOIN操作进行.下面我们介绍有关并行JOIN算法的研究结果.

1.1 基于嵌套循环的并行JOIN算法

由于嵌套循环JOIN算法必须存取两个JOIN关系的笛卡尔乘积,在顺序计算机环境中,它一直被认为是效率最低的JOIN算法.然而,嵌套循环JOIN算法很容易并行化[1],并且可以通过附加的计算减少多处理机间的通讯开销.文献[2]给出了一个典型的并行嵌套循环JOIN算法(简称PNLJ算法):

输入:关系 R 和 S,处理结点个数 N.

输出:关系 R 和 S 的JOIN结果.

(1)选择小关系,比如 S;

(2)把 S 均匀地分布到多个处理结点,设 S_i 是 S 在结点 i 上的子集合;

(3)FOR $i=1$ TO N DO (并行地)

以流水线方式向 N 个结点发送 R 的元组;

结点 i 在 S_i 中查找在JOIN属性上与所收到的 R 的元组匹配的元组,

进行 S_i 与 R 的JOIN;

---

* 本文1993—07—11定稿

本项目是国家自然科学基金资助项目,并受到国家教委优秀青年教师基金,黑龙江省科委和哈尔滨市科委的资助.

作者李建中,44岁,教授,主要研究领域为并行软件系统与并行数据库系统.

本文通讯联系人:李建中,哈尔滨 150080,黑龙江大学信息技术研究所

(4)输出 R 和 S 的 JOIN 结果.

文献[1]的性能分析表明,在具有大量处理结点的系统中或在 R 和 S 的大小差别很大的情况下,PNLJ 算法的效率远高于其它并行 JOIN 算法.

1.2 基于 SORT－MERGE 的并行 JOIN 算法

基于 SORT－MERGE 的并行 JOIN 算法(简称 PSMJ 算法).由两个阶段组成,排序阶段和 JOIN 阶段.在排序阶段,它按照 JOIN 属性的值排序每个 JOIN 关系;在 JOIN 阶段,使用合并算法完成两个排序关系的 JOIN.文献[3,4]给出了一个具有代表性的 PSMJ 算法:

输入:关系 R 和 S,处理结点个数 N,HASH 函数 H.

输出:关系 R 和 S 的 JOIN 结果.

(1)使用 H 在 N 个处理结点间分布 R 和 S 的元组,

设 S_i 和 R_i 是 S 和 R 在结点 i 上的子集;

(2)FOR $i=1$ TO N DO (并行地)

处理结点 i 排序 R_i 和 S_i;

(3)FOR $i=1$ TO N DO (并行地)

结点 i 使用合并算法完成 R_i 与 S_i 的 JOIN;

(4)输出 R 和 S 的 JOIN 结果.

理论和实验结果说明,当两个 JOIN 关系的元组数相差较小时,PSMJ 算法的效率高于 PNLJ 算法.

1.3 基于 HASH 的并行 JOIN 算法

基于 HASH 的 JOIN 算法很容易并行化[3].它通过一个定义在 JOIN 属性上的 HASH 函数把 JOIN 关系分解为 N 个子集合,然后使用 N 个处理机并行地完成 JOIN 操作.以下,我们简称并行 HASH－JOIN 算法为 PHJ 算法.如果 HASH 函数能够把 JOIN 关系均匀地分布为 N 个子集合,则 PHJ 算法可以具有线性时间复杂性.这里我们介绍两个具有代表性的 PHJ 算法.1983 年,Kitsyregawa 提出了第一个 PHJ 算法[5],称为 GRACE－HASH－JOIN 算法,其定义如下:

输入:关系 R 和 S,HASH 函数 H_1 和 H_2,处理结点数 N,子集合数 M.

输出:关系 R 和 S 的 JOIN 结果.

(1)使用 H_1 把 R 划分为 M 个子集合,HASH 值为 i 的元组送入子集合 R_i,并存储到所有可用的磁盘;

(2)使用 H_1 把 S 划分为 M 个子集合,HASH 值为 i 的元组送入子集合 S_i,并存储到所有可用的磁盘;

(3)FOR $i=1$ TO M DO (并行地)

使用 H_2 把 R_i 分布到 N 个处理结点,在每个结点内存中建立 R_i 元组的 HASH 表;

使用 H_2 以流水线方式向 N 个处理结点发送 S_i 的元组;

FOR $j=1$ TO N DO (并行地)

结点 j 用收到的 S_i 的元组匹配 R_i 的 HASH 表,进行 S_i 和 R_i 的 JOIN;

(4)输出 R 和 S 的 JOIN 结果.

算法中的 M 应该充分大,以减少 R 的子集合对应的 HASH 表超过可用内存容量的概

率. Schneder 和 DeWitt 改进了 GRACE－HASH－JOIN 算法，提出了一个称为 HYBRID－HASH－JOIN 的 PHSJ 算法[3]. 在这个算法中，假定 N 个处理结点中 L 个处理结点无磁盘. 算法定义如下：

输入：关系 R 和 S，HASH 函数 H_1、H_2 和 H_3，处理结点数 N，子集合数 M，无磁盘结点数 L.

输出：关系 R 和 S 的 JOIN 结果.

(1)使用 H_1 把 R 划分为 M 个子集合 R_1、…、R_M；

使用 H_3 分布 R_1 到 L 个无磁盘处理结点，并在每个结点内存中建立 R_1 元组的 HASH 表；

R_2、…、R_M 存入所有可用磁盘；

(2)使用 H_1 把 S 划分为 M 个子集合 S_1、…、S_M；

S_2、…、S_M 存入所有可用磁盘；

使用 H_3 以流水线方式向 L 个无磁盘处理结点发送 S_1 的元组；

FOR　i=1　TO　L　DO (并行地)

　　结点 i 用收到的 S_1 的元组匹配 R_1 的 HASH 表，进行 S_1 和 R_1 的 JOIN；

(3)FOR　i=2　TO　M　DO (并行地)

使用 H_2 把 R_i 分布到 N 个处理结点，在每个结点内存中建立 HASH 表；

使用 H_2 以流水线方式向 N 个处理结点发送 S_i 的元组；

FOR　j=1　TO　N　DO (并行地)

　　结点 j 用收到的 S_i 的元组匹配 R_i 的 HASH 表，进行 S_i 和 R_i 的 JOIN；

(4)输出 R 和 S 的 JOIN 结果.

HYBRID－HASH－JOIN 算法实现了 JOIN 关系 R 和 S 的划分与其第一子集 R_1 和 S_1 的 JOIN 操作并行运行，提高了运行速度. 文献[3]在具有每秒 80M 字节通讯速度的 TOKEN－RING 网络环境下实现了这个算法. 文献[6]在 CUBE 和 RING 多处理机系统中实现了这个算法.

1.4　数据分布的均匀性与并行 JOIN 算法

很多 JOIN 算法的研究都假定 JOIN 关系在 JOIN 属性上均匀分布[6]. 这种均匀性假定是不实际的. 很多研究已经表明，在现实数据库中，数据的分布常常是不均匀的[7－9]. 这种数据分布不均匀性称为数据偏斜. 文献[7]说明，数据偏斜对并行 JOIN 算法的效率影响很大. 文献[8]说明，处理结点越多，数据偏斜对并行 JOIN 算法效率的影响越大. 数据偏斜对 PHJ 算法的性能有两方面的影响. 一方面它可使 HASH 桶溢出，即其所需存储空间超过可用内存容量. 溢出桶不得不被再划分，增加了额外的 I/O 和其它系统开销，降低了 JOIN 算法的效率. 另一方面它可能使 JOIN 关系的数据在处理结点间分布偏斜，引起工作负载的不平衡，降低 JOIN 算法的效率. 最近几年，出现了一些克服数据偏斜影响的并行 JOIN 算法.

文献[8]提出一个克服数据偏斜影响的 PSMJ 算法. 这个算法在并行 SORT－MERGE 算法的排序阶段之后增加了一个调度阶段，把排序阶段的结果作为输入，在多处理结点间再分布工作负载，使各处理结点的工作负载平衡.

文献[10]提出了另一个克服数据偏斜影响的 PHJ 算法. 这个算法首先扫描 JOIN 关系，

获得它们的数据分布信息.然后,使用这些信息进行数据的再划分,以保证工作负载分布均匀.

文献[11]提出了一个称为动态HYBRID－GRACE－HASH－JOIN的PHJ算法.这个算法采用了动态调整HASH桶大小的方法避免HASH桶溢出问题.它的基本思想是:使用HASH函数把JOIN关系划分为许多小的HASH桶.每个桶的大小不超过一个处理结点的可用内存容量.当在多处理结点的主存中建立HASH表时,只要主存还有空余空间,就向主存反复加载更多的桶,直至所有处理结点的主存用完为止.为了提高I/O效率,如果某些桶的大小低于一个磁盘页的容量,则把它们组合成一些与磁盘页容量近似的桶.

文献[12]提出了第四个PHSJ算法,称为SPREADING－PARALLEL－HASH－JOIN算法(SPHJ),分四个阶段解决工作负载不平衡问题:

(1)划分阶段:并行地把JOIN关系分成多个HASH桶,每个桶都通过OMEGA连接网络均匀地分布到所有可用的处理结点;

(2)桶调整阶段:调整桶的大小,使每个桶大小近似等于可用内存的容量,避免桶溢出;

(3)划分调整阶段:首先,根据桶的大小,把所有的桶分为N组,使各组具有近似相等的数据量,这里,N是可用处理结点个数;然后,按照Round－Robin方式确定在可用处理结点间分配桶组的方案,但不移动数据;

(4)JOIN阶段:各处理结点按照桶组分配方案并行地从其它结点逐一地搜集分配到它的HASH桶,进行JOIN处理,直至所有桶处理结束.这个算法的问题是紧密依赖于OMEGA连接网络.文献[13]解决了这个问题.它在划分阶段使用软件方法实现了每个桶在多处理结点间的均匀分布.基于SPHJ算法,文献[14]提出了一个在具有共享存储器并行计算机上运行的JOIN算法.这个算法可以有效地处理具有有限数据偏斜关系的JOIN.

文献[15]利用共享虚拟存储器并行结构的特点,扩展了HYBRID－HASH－JOIN算法,使用共享工作负载的方法处理数据偏斜带来的工作负载不平衡问题.它的基本思想是:当一个处理结点结束运行时,检查是否有其它处理结点仍在处理同一个JOIN操作.如果有,则挑选一个工作负载最重的处理结点,将其负载分出一部分交给空闲的处理结点.由于共享虚拟存储器并行结构非常适用于这种工作负载的动态再分配,这个算法具有较高效率.

1.5 数据的初始划分与并行JOIN算法

在并行关系数据库系统中,当一个关系初始加载时,数据库管理系统通常都要使用某种方法将其划分为多个子集合,分布到多个处理结点.关系的初始数据分布对以后该关系上的数据操作有很大的影响.如果在设计一个并行数据操作算法时能够充分利用初始数据分布的特点,其结果算法将会十分有效.但是,上述并行JOIN算法都没有利用关系的初始数据划分特点.它们在进行JOIN处理之前,不管其初始数据分布是否已经可以有效地支持JOIN操作的处理,都要重新划分JOIN关系,引起了不必要的I/O和其它系统开销以及数据偏斜问题.1992年,本文作者曾经以CMD数据分布方法为基础,设计分析了并行多维区域查询操作算法[16].理论分析和模拟实验都表明,这个算法具有相当高的效率.同年,本文作者还在CMD数据分布的基础上,提出了一个充分利用关系的初始数据分布特点的并行JOIN算法,称为DA－JOIN算法[17].DA－JOIN算法具有以下几个特点:(1)利用CMD关系部分排序的特点,剔除两个JOIN关系中不可能进行JOIN的部分,极大地减少了I/O和

通讯时间，并且打破了 JOIN 操作至少存取整个操作关系一遍的复杂性下界；(2)避免了数据偏斜问题；(3)具有 HYBRID－HASH－JOIN 算法的所有优点，但要求远小于它的 I/O 工作量. 理论分析和在 CM5 机上的实验结果都表明，DA－JOIN 算法的效率高于其它并行 JOIN 算法.

2　查询优化技术

查询优化不仅是顺序数据库系统的重要组成部分，也是并行数据库系统的重要组成部分. 目前，无论是在顺序数据库系统领域还是在并行数据库系统领域，查询优化的研究主要围绕着具有多个 JOIN 操作的复杂关系数据库查询(简称 MJ 查询)的优化问题进行.

并行数据库查询优化问题与顺序数据库查询问题不同. 在顺序数据库系统中，给定一个查询 Q，查询优化算法只需找到 Q 的一个具有最小工作量的执行计划. 这样的计划必然具有最快的响应时间. 这是因为在具有单处理机的计算机系统中，一个计算任务的响应时间与这个任务的工作量成正比. 在并行数据库系统中，情况就不同了. Q 的具有最小工作量的执行计划可能具有很强的固有顺序性，难以并行化. 在并行数据库系统中，查询优化的目标是寻找 Q 的具有最小响应时间的执行计划，执行计划的工作量不是第一重要的. 于是，在并行数据库系统中，我们需要新的查询处理算法和新的查询优化技术. 并行数据库查询优化的研究分为两个方面：查询执行计划(简称查询计划)模型和查询优化算法. 查询优化算法与查询计划模型密切相关. 下边我们按查询计划模型分类介绍并行数据库查询优化方面的主要研究成果.

2.1　基于左线性树的查询优化算法

本节首先介绍查询树模型，然后讨论基于左线性树的查询优化算法.

2.1.1　查询树模型

1990 年，Schneder 等研究了查询树模型[18]，提出了左线性树(left－deep trees)、右线性树(Right－deep trees)和浓密树(Bushy Trees)的概念. 一个 MJ 查询可以表示为一个查询树 $G=(V,E)$，其中，V 是结点集合，E 是边集合. 每个叶子结点表示一个关系. 每个内结点表示一个 JOIN 操作，同时表示这个 JOIN 操作的结果关系. 每条边表示一个 JOIN 操作，JOIN 谓词也可以由边来表示. 显然，MJ 查询的查询树是一个二叉树. 查询树可以分为三类：左线性树、右线性树和浓密树. 左线性树是一个特殊的查询树，每个内结点(包括根结点)的右子结点必须是一个 JOIN 关系，左子结点是一个内结点或一个 JOIN 关系. 右线性树也是一个特殊的查询树，每个内结点(包括根结点)的左子结点必须是一个 JOIN 关系，右子结点是一个内结点或一个 JOIN 关系. 除了左线性树和右线性树以外的查询树皆称为浓密树.

2.1.2　基于左线性树的查询优化算法

文献[18]给出了一种以左线性树和 HASH－JOIN 算法为基础的 MJ 查询的优化方法，以下简称 LDT 方法. LDT 方法把两个 JOIN 关系分为内关系和外关系，把 HASH－JOIN 分为 BUILD 和 PROBE 两个基本操作. BUILD 操作扫描内关系，建立 HASH 表；PROBE 操作扫描外关系，搜索匹配 HASH 表，完成 JOIN 操作. LDT 方法使用数据相关图的概念模拟 HASH－JOIN 的处理过程，确定左线性树的优化执行计划. 图 1 给出了具有 N 个 JOIN

操作的MJ查询的左线性树及其数据相关图.图中B_i和P_i分别表示第i个JOIN操作的BUILD和PROBE操作.第i个原始JOIN关系被表示成S_i,即扫描(SCAN)第i个关系的操作.虚线框内的操作是可以并行执行的操作,实线连接的两个虚线框必须顺序执行,箭头所指框必须在箭尾所连框之后执行.

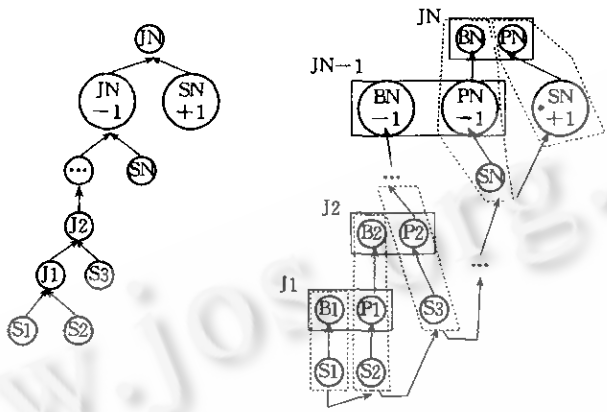

图1 具有N个JOIN操作的MJ查询的左线性树及其数据操作相关图

给定MJ查询Q,Q的每个左线性树都包含唯一一个具有最高并行性的查询计划,称为Q的左线性树优化并行查询计划.图1中的左线性树及其数据相关图给出的优化并行查询计划如下:

1. 并行执行:S_1,B_1
2. 并行执行:S_2,P_1,B_2
3. 并行执行:S_3,P_2,B_3

……

N. 并行执行:S_N,P_{N-1},B_N

N+1. 并行执行:S_{N+1},P_N.

使用文献[18]的方法,我们可以得到如下的MJ查询优化算法(简称LDT算法):

1. 搜索给定MJ查询的左线性树空间,选择具有最小响应时间的优化左线性树T;
2. 由T产生数据相关图DG;
3. 根据DG产生优化并行查询执行计划P;
4. 运行优化并行查询计划P.

从图1可以看到,在MJ查询的左线性树优化查询计划中,同时可有两个JOIN操作并行执行,因此需要两个HASH表的存储空间.除了J_1的HASH表以外,所有其它HASH表都是由中间JOIN操作的结果关系建立的.这些中间结果关系的大小是很难预测的.既使能够预测,在多用户环境下优化处理程序也很难知道查询计划执行时可用空间的大小.如果有足够的主存空间,问题很简单,只需为HASH表分配充分大的空间.如果可用主存空间有限,HASH表大小的确定则成为影响并行查询执行计划效率的大问题.解决这个问题的一种简单方法是令查询计划调度程序根据系统中可用存储空间的大小动态地调整HASH桶的规模.文献[19]提出了另一种解决方法:优化处理程序产生多个查询计划,系统运行时根据系统中可用存储空间的大小选择合适的查询计划执行.一个类似的机构已经在STAR-

BURST 系统上实现[20].

LDT 算法存在 3 个问题：(1)由于仅考虑了 HASH—JOIN 算法和 MJ 查询的左线性树查询计划，难以保证产生高效率的查询计划；(2)由于至多有两个 JOIN 操作可以并行执行，数据操作间的并行性得不到充分的发挥；(3)MJ 查询的左线性树空间搜索算法需要进一步研究.

1992 年，Ganguly 等以左线性树为基础，提出了一种支持多种 JOIN 算法的查询优化方法[21]，研究了并行数据库查询优化的几个基本问题，提出了表示查询计划的操作树概念，建立了以操作树为基础的两种并行查询计划复杂性模型：无资源竞争的响应时间模型和有资源竞争的响应时间模型. 他们以有资源竞争的响应时间模型为优化的目标，给出了一个产生优化左线性树查询计划的动态规划算法. 这个算法能够支持多种 JOIN 算法，而且效率高于 LDT 算法. 但是，由于它仅考虑了 MJ 查询的左线性树查询计划，仍具有 LDT 算法的缺点.

2.2　基于右线性树的查询优化算法

显然，由于 MJ 查询的左线性树优化并行查询计划最多只允许两个 JOIN 操作并行执行，这种方法不能充分发挥数据操作间的并行性. 为此，Schneder 等研究了基于右线性树的 MJ 查询优化问题，给出了使用数据相关图概念确定右线性树的优化执行计划的方法[18]. 图 2给出了具有 N 个 JOIN 操作的 MJ 查询的右线性树表示及其数据相关图.

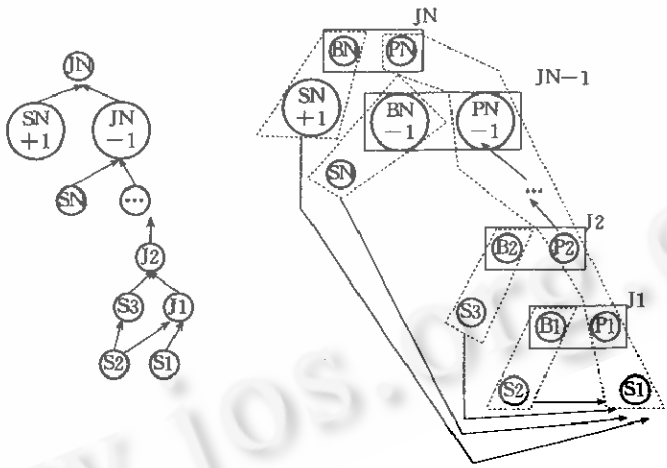

图2　右线性树及其数据相关图

下面是相应的具有最高并行性的执行计划，称为右线性树优化并行查询计划：

1. 并行执行：$S_1, B_1, S_3, B_2, \cdots, S_{N+1}, B_N$

2. SCAN $S_2, P_1, P_2, \cdots, P_N$.

使用文献[18]给出的方法，我们可以得到基于右线性树的查询优化算法(简称 RDT 算法). RDT 算法类似于 LDT 算法. RDT 算法产生的查询执行计划具有相当高的并行性，所有 N 个 JOIN 操作皆可并行执行. 然而，这样的执行计划要求很大的存储空间来存储所有 N 个 JOIN 操作的 HASH 表. 如果具有足够的主存空间，这样的执行计划具有很高的效率. 当可用主存空间有限时，存储空间分配将成为严重的问题. 目前，已经出现几种解决这个问题的方法.

一种方法称为片段或右线性树调度方法[22]. 这种方法首先使用优化程序或查询计划调度执行程序,把右线性树分成多个不相交的可执行片段,使得每个片段的存储要求不超过可用的存储空间. 然后,使用右线性树和数据操作相关图方法为每个片段确定具有最高并行性的执行计划. 最后,顺序地执行各个片段. 这种方法需要把每个片段的执行结果暂存到磁盘上,作为下一个片段执行的输入.

另一种方法是一个动态自底向上的调度方法. 它的基本思想是:令查询计划调度执行程序动态地检查是否具有足够的存储空间并行执行多个 JOIN 操作,保证在可用的主存空间上并行执行尽可能多的 JOIN 操作.

文献[18]提出了一个 HYBRID 调度方法. 它的基本思想是:在 BUILD 阶段,每个 JOIN 关系被分成两部分,一部分进入内存,建立 HASH 表,另一部分留驻外存,待以后处理;在 PROBE 阶段,每个 JOIN 操作的结果也被分成两部分,一部分直接用来进行下一个 JOIN 操作的 PROBE 处理,另一部分存入外存,待以后处理.

RDT 算法的效率高于 LDT 算法,但仍存在类似于 LDT 算法的问题.

2.3 基于片段式右线性树的查询优化方法

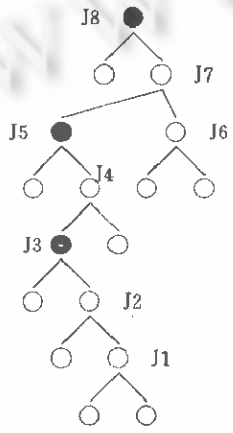

图3 一个SDR树的实现

对左线性树和右线性树查询计划的性能进行比较,我们可以发现,右线性树查询计划优于左线性树查询计划. 然而,右线性树查询计划仍然很受限制,除了要求内存量大以外,还可能排斥很多具有更高性能的查询计划. 在这方面,浓密树要比右线性树优越,但对应的查询计划空间非常大,相应的优化算法的开销也相当大. 为了克服右线性树和浓密树的缺点,Chen 等提出了一种界于右线性树和浓密树之间的片段式右线性树(简称 SRD 树)模型,并以 HASH－JOIN 为基础给出了基于 SRD 树的查询优化算法[23]. SRD 树是一种特殊的浓密树,由一组右线性树连接而成,每个右线性树称为一个片段,每个片段中的一个 JOIN 操作称为一个阶段. 图 3 给出了一个 SRD 树的实现.

图中的实心结点表示各片段的根结点,未加标记的结点表示关系,J_i 是第 i 个 JOIN 操作. Chen 等给出了如下的 SRD 树调度执行策略:以片段为单位顺序执行,每个片段按右线性树的方式并行执行,其结果作为下一个片段的输入关系. 一个片段未结束之前,它的上一级片段(即其根结点所连接的片段)不能执行. 他们还提出了一种启发式优化 SRD 树生成算法. 基于 SRD 树的查询优化算法的效率高于前面介绍的算法. 但是,它也不能保证产生高效率的查询计划.

2.4 基于浓密树的查询优化算法

1991 年,Lu 等提出了基于浓密树的 MJ 查询优化算法[24]. 他们把查询计划分为同步和非同步两类. 在同步查询计划中,一个 MJ 查询的处理过程划分为多个同步执行阶段. 各个同步执行阶段顺序执行. 在每一同步执行阶段,多个 JOIN 操作并行执行. 其它的查询计划称为非同步执行计划. 文献[24]以同步执行计划为目标,提出了三种产生优化同步执行计划的启发式 MJ 查询优化算法. 第一个算法称为 GP 算法,是一种贪心算法. 算法的核心是同

步执行段的划分. 这个算法循环地为每一同步执行阶段选择尽量多的可并行执行的 JOIN 操作. 为了减少 GP 算法的时间复杂性，他们提出了另外两个启发式贪心算法. 一个算法称为 GP_t，仅产生如下类型的计划：只有第一同步执阶段并行执行多个 JOIN，其余同步执行段仅执行一个 JOIN 操作. 另一个算法称为 GP_p. GP_p 简化了 GP_t 查询计划的代价模型，其余皆与 GP_t 相同. 他们的算法具有 4 个问题. 一是丢弃了同步执行阶段之间的流水线并行性. 二是增加了读写中间结果的 I/O 开销. 三是由于同步的原因，某些较早完成 JOIN 任务的处理结点必须空闲等待，降低了处理结点的利用率. 四是由于仅考虑了同步查询计划并使用了可能排除高效查询计划的启发规则，难以保证产生高效率的查询计划.

2.5　基于操作森林的查询优化算法

我们在文献[25]中提出了一个以操作森林为基础的 SELECT－PROJECT－JOIN 查询(简称 SPJ 查询)的并行查询优化算法. 这个算法比上面介绍的算法更具有一般性，因为它不仅支持 MJ 查询也支持应用程序最经常使用的 SPJ 查询. SPJ 查询的操作森林由多个操作树组成. 这里的操作树类似于文献[21]中的操作树，只是操作集合不同. 我们的操作集合包括下列基本操作：

Scan：扫描一个关系，对应于 Select、Project、数据再分布操作；

Sort：对输入关系元组集合进行排序；

Hash：由输入关系元组集合建立 Hash 表；

Probe：使用输入关系元组集合搜索匹配 Hash 表，输出匹配的元组；

Merge：合并两个排序的输入关系元组集合；

Nested－loops：按照给定规则比较两个关系的元组，输出匹配的元组.

基于操作森林的查询优化算法由三部分组成：(1)生成操作森林；(2)为每个操作树分配系统资源. (3)调度森林中各树并行运行.

给定一个 SPJ 查询，操作森林生成算法如下：

(1)生成 SPJ 查询的语法树[31]；

(2)转化 SPJ 查询的语法树为操作树：

a. 在每个关系与其父结点之间插入一个 scan；

b. 在直接相连的每对 JOIN 结点间插入一个 Scan(表示数据分布)；

c. 选择实现 JOIN 操作的算法，把每个 JOIN 结点用一个由基本操作组成的树代替；

(3)在操作树上标记顺序执行边：若 B 是 A 的父结点，B 必须在 A 完成之后才能开始运行，则边(A,B)标记“M”，即顺序执行；

(4)删除操作树中标记“M”的边，形成操作森林.

给定一个操作森林，资源分配算法如下(书架分配法)：

(1)根据数据相关性，对操作森林中所有操作树进行拓扑排序，确定操作树的执行顺序图，执行顺序图中的结点是一个操作树；

(2)计算每个操作树 t 的工作量 W(t)，即 t 在单处理机上的执行时间；

(3)在执行顺序图中确定关键路径，即最长路径；

(4)从书架的第一格开始，把关键路径的各结点按执行顺序图由叶到根的顺序分别存入不同的格子；

(5)按下述规则把执行顺序图中其余结点分配到各格子中：

a. 自底向上；

b. 只分配执行顺序图的叶结点；

c. 每个结点所分格子必须低于父结点所在格子；

d. 如果多个同级结点可分配，先分配具有大工作量的结点；

e. 当一个结点可分到多个格子时，分到具有最小总工作量的格子；

f. 当一个结点被分配到一个格子后，从顺序执行图中删除该结点及其相关的边；

(6)在每个格子中的所有操作树之间分配系统的所有处理结点：

a. 先为每个操作树分配一个或两个处理结点，(具有流水线的树分两个处理结点)；

b. 使用贪心法把剩余处理结点按操作树间的工作量比例分配给各操作树. 资源分配结束后，各子操作树的运行很简单，只需从第一格开始顺序运行各格子的所有树，每个格子中操作树并行运行.

2.6 有关查询优化的其它研究成果

2.6.1 一般化的并行数据库查询表示模型

本文作者在文献[26]中研究了查询优化的几个基本问题，以多重加权树为基础，提出了一种一般化的查询计划及其复杂性模型. 下边的定义 2.1 给出了这个模型的严格定义.

定义 2.1. 设 N 是处理结点数，M 是存储器容量，OP 是数据操作集合，MT 是实现 OP 中所有数据操作的全部算法，R 是关系集合. 一个并行查询计划是一个多重加权树$((V,E),F_1,F_2)$，其中，(V,E)是一个树，$V=OP\cup R$，$E=\{(X,Y)\mid X,Y\in OP$ 且 Y 的输出是 X 的输入，或 $X\in OP$ 且 $Y\in R$，Y 是 X 的操作关系$\}$，F_1 和 F_2 是函数，F_1：$OP\to\{1,\cdots,N\}\times\{1,\cdots,M\}\times MT$，$F_2$：$(E-\{(X,Y)\mid Y\in R\})\to\{0,1,\cdots,M\}\times\{P,S\}$.

从定义 2.1 不难看出，这个查询计划表示模型具有如下特点：

1. 支持多种数据操作(由集合 OP 定义)；

2. 支持各种数据操作的实现算法(由集合 MT 定义)；

3. 能够表示处理结点、存储器和实现算法的分配(由函数 F_1 表示)；

4. 能够表示关系数据库查询的三种固有并行性，即单数据操作的并行执行(由函数值 $F_1(op)=(n,m)(m>1)$表示)、多个数据操作的流水线式并行执行(由函数值 $F_2(e)=(X,P)$表示)和数据操作间的独立并行执行(由多重加权树的不同路径上的内结点表示).

文献[26]还给出了一种查询计划的复杂性模型. 这个模型使用响应时间作为查询计划的复杂性量度，由两部分组成. 一部分是单个数据操作的响应时间. 另一部分是整个查询计划的响应时间. 单个数据操作的响应时间是一个多变量函数 $RT(O,A,N,M,S,D,ST,SR)$，其中，O 是数据操作，A 是 O 所使用的算法，N、M 和 D 分别是分配给 O 的处理结点数、存储器容量和磁盘数，S 是操作数据库的大小，ST 是并行计算机结构，SR 是 O 的结果关系大小. 查询计划的响应时间 $RT(PLAN)$可使用单个数据操作的响应时间沿查询树自底向上地求得，限于篇幅，在此不详述，参见文献[26].

2.6.2 查询优化算法

1992 年，Chen 等提出了以单处理机上的优化查询计划为基础的 MJ 查询优化方法[27]. 他们的方法首先产生一个优化的顺序查询执行计划，然后并行化这个计划，产生一个高效率

的并行查询执行计划.但这个计划不一定是优化的.

1992 年,Hong 等提出了一种以共享内存为并行结构以资源利用率为目标以 SORT－MERGE－JOIN 算法为基础的查询优化算法[28].这种算法把查询中具有高 I/O 负载的操作和具有高 CPU 负载的操作组合成并行执行操作对,令其并行执行,产生充分利用系统资源的查询计划.在查询计划执行期间,这种算法动态地调整查询计划的并行性,使查询计划始终在保持资源利用率最高的条件下运行.

1993 年,Lo 等研究了 MJ 查询的流水线式执行计划的处理结点分配问题,提出了 3 个为顺序查询计划分配处理结点、实现查询并行化的方法[29].

3　需要进一步研究的问题

并行数据库系统是一个新兴的研究领域,虽然已经取得了一些成果,仍有大量的问题需要解决.本节简要地讨论需要进一步研究的问题.

3.1　并行数据操作算法.

我们已经看到,并行数据操作算法的研究一直围绕着 JOIN 操作进行.而且,这些工作多以 HYPERCUB 并行连接结构为基础.以其它常用的并行连接结构为基础的并行 JOIN 算法需要进一步研究.除了 JOIN 操作外,数据库管理系统还有很多其它数据操作,特别是复杂的数据操作,如聚集、统计等方面的操作.这些数据操作的并行算法的研究还很少,需要深入开展.充分利用初始数据分布特点的并行数据操作算法的设计与分析也是一个重要的研究课题.

3.2　并行数据库的查询优化.

到目前为止,还没有一个并行查询优化算法能够保证产生高效率查询计划.我们需要开展以下几个方面的工作:

a. 具有强表达能力的并行查询计划表示模型的研究.

b. 并行数据操作算法的多变量复杂性模型的研究,即各种数据操作在不同的并行结构、不同的实现算法、不同的处理结点数、不同的存储空间等条件下的复杂性模型.这是并行查询计划的基础.

c. 以并行数据操作算法的多变量复杂性模型为基础的查询计划的复杂性模型的研究.

d. 具有产生高效率查询计划能力的并行查询优化算法的研究,特别是减少查询计划搜索空间而不丢失高效率查询计划的启发式规则的研究.

e. 抗数据偏斜影响的并行数据库查询优化方法的研究.

f. 多用户环境下的并行查询优化的研究.

g. 具有混合工作负载的多查询调度算法的研究.

h. 查询优化算法本身的并行化研究.

3.3　并行数据库的物理设计

并行数据库物理设计的研究包括以下两个方面.一是数据库划分方法的研究.这种方法应该有效地支持数据库的基本数据操作.虽然目前已经出现了一些数据库分布方法,但还很不够,特别是适合于动态数据库的划分方法,如动态多维数据分布方法.二是数据库设计工具的研究.这样的工具以给定的数据库及各种参数和系统环境为输入,确定数据库文件、索

引等数据集合在多处理结点间分布的策略.

3.4 数据加载和再组织

大型数据库的数据加载、再组织和备份可能需要数百小时.显然,在这种情况下,并行性是十分必要的.很遗憾,除了我们曾研究提出了几个并行GRID文件的并行加载和再组织算法外[30],还没有其它的工作出现.并行数据加载和再组织是一个需要深入研究的领域.在这个领域中有一个非常困难问题,即如何在数据加载、再组织和后援备份时仍能保证数据库的可操作性.

参考文献

1 Valduriez P, Gardarin G. Join and semijion algorithms for a multiprocessor database machine. ACM Trans. on Database Systems, 1984,9(1): 133—161.

2 Boral H *et al*. Join on a cube: analysis, simulation and implementation. In: Kitsuregawa M, Tanaka H eds. Database Machines and Knowledge Base Machines, Boston: Kluwer, 1988: 61—74.

3 Schneider D A, DeWitt D J. A performance evaluation of parallel in algorithms in a shared—nothing multiprocessor environment. In: Maier D ed. Proc. of ACM SIGMOD'89, USA, 1989, Baltimore: ACM Press, 1989:110—121.

4 DeWitt D J, Gerber R. Multiprocessor hash—based join algorithms. In: Pirotte A, Vassiliou Y eds. Proceedings of VLDB'85, Stockholm, 1985, San Mateo: Morgan kaufmann Publishers, Inc., 1985:151—164.

5 Kitsuregawa M, Tanaka H, Moto—oka T. Application of hash to data base machine and its architecture. New Generation Computing, 1983,1(1):25—39.

6 Omiecinski E R, Lin E T. Hash—based and index—based join algorithms for cube and ring connected multicomputers. IEEE Trans. on Knowledge and Data Eng., 1989,1(3):329—343.

7 Lakshmi M S, Yu P S. Effectiveness of parallel joins. IEEE Trans. on Knowledge and Data Eng., 1990,2(4):410—424.

8 Wolf J L, Dias D M, Yu P S. An effective algorithm for parallizing sort merge joins in the presence of data skew. Proceedings of the 2nd International Symp. on Databases in Parallel and Distributed Systems, USA, 1990.

9 Lynch C A. Selectivity estimation and query optimization in large databases with highly skewed distributions of column values. In: Bancilhon F, DeWitt D J eds. Proceedings of VLDB'88, Los Angeles, 1988, San Mateo: Morgan kaufmann Publishers, Inc., 1988:240—251.

10 Wolf J L *et al*. An effective algorithm for parallel hash joins in the presence of data skew. Technique Report RC 15510, Watson: IBM T. J. Watson Research Center, 1990.

11 Kitsuregawa M, Nakayama M, Takagi M. The effect of bucket size tuning in the dynamic hybrid GRACE hash join method. In: Peter M G, Wiederhold A G eds. Proceedings of VLDB'89, Amsterdam, 1989, San Mateo: Morgan kaufmann Publishers, Inc., 1989: 257—266.

12 Kitsuregawa M, Ogawa Y. Bucket spreading parallel hash: a new pobust, parallel hash join method for data skew in the super database computer(SDC). In: McLeod D, Sacks—Davis R, Schek H eds. Proceedings of VLDB'90, Brisbane, 1990, Palo Alto: Morgan kaufmann Publishers, Inc., 1990:210—221.

13 Hua K A, Lee C. Handling data skew in multiprocessor database computer systems using partition tuning. In: Lohman G M, Sernadas A, Camps R eds. Proc. of VLDB'91, Barcelona, 1991, San Mateo: Morgan kaufmann Publishers, Inc.,1991:525—536.

14 Omiecinski E. Performance analysis of a load balancing hash—join algorithm for a shared memory multiprocessor. In: Lohman G M, Sernadas A, Camps R eds. Proc. of VLDB'91, Barcelona, 1991, San Mateo: Morgan kaufmann Publishers, Inc., 1991:375—386.

15 Shatdal A, Naughton J F. Using shared virtual memory for parallel join processing. In: Buneman P, Jajodia S eds. Proceedings of ACM SIGMOD'93, Washington DC, 1993, Baltimore: ACM Press, 1993:119—128.

16 Li Jianzhong. Range query procession in multidisk systems. Journal of Computer Science and Technology, 1992, 7(4):316—327.

17 Niccum T M, Li Jianzhong. DA—Joins: declustering aware parallel join algorithms. Technical Report TR 92—71,Minneapolis: Department of Computer Science, University of Minnesota, USA, 1992.

18 Schneider D, DeWitt D J. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In: Leod D, Sacks—Davis R, Schek H eds. Proc. of VLDB'90, Brisbane, 1990, Palo Alto: Morgan kaufmann Publishers, Inc., 1990:469—480.

19 Graefe G, Ward K. Dynamic query evaluation plans. In: Maier D ed. Proceedings of ACM SIGMOD'89, USA,

1989, Baltimore: ACM Press, 1989:358—366.
20 Haas L *et al*. Extensible query processing in starburst. In: Maier D ed. Proceedings of ACM SIGMOD'89, USA, 1989, Baltimore: ACM Press, 1989:377—388.
21 Ganguly S, Hasan W, Krishnamurthy R. Query optimization for parallel execution. In: Stonebraker M ed. Proceedings of ACM SIGMOD'92, San Diego CA, 1992, Baltimore: ACM Press, 1992:9—18.
22 Stonebraker M, Aoki P, Seltzer M. Parallelism in XPRS. Memorandum, No. UCB/ERL M89/16, Feb. 1989.
23 Chen M S *et al*. Using segmented right—deep tree for the execution of pipelined hash joins. In: Yuan Liyan ed. Proceedings of VLDB'92, VanCouver, 1992, San Mateo: Morgan kaufmann Publishers, Inc., 1992:15—26.
24 Lu H, Shan M C, Tan K L. Optimization of multi—way join queries for parallel execution. In: Lohman G M, Sernadas A, Camps R eds. Proceedings of VLDB'91, Barcelona, 1991, San Mateo: Morgan kaufmann Publishers, Inc., 1991:549—560.
25 Nicumm T M, Srivastava J, Li Jianzhong. A tree—decomposition approach to the parallel execution of relational query plans. AHPCRC Technique Report No. 93—019, Minneapolis: Army High Performance Computing Research Center, USA, 1993.
26 李建中. 并行关系数据库系统的并行查询计划模型. 第 10 届中国数据库学术会议论文集. 沈阳：东北工学院出版社, 1992:58—64.
27 Chen M S, Yu P S, Wu K L. Scheduling and processor allocation for parallel execution of multi—join queries. In: IEEE Computer Society ed. Proceedings of the 8th International Conf. on Data Engineering, Tempe, Arizona, 1992, Los Alamitos: IEEE Computer Society Press, 1992:58—67.
28 Hong W. Exploiting inter—operator parallelism in XPRS. In: Stonebraker M ed. Proceedings of ACM SIGMOD'92, San Diego CA, 1992, Baltimore: ACM Press, 1992:19—28.
29 Lo M L *et al*. On optimal processor allocation to support pipelined hash joins. In: Buneman P, Jajodia S eds. Proceedings of ACM SIGMOD'93, Washington DC, 1993, Baltimore: ACM Press, 1993:69—78.
30 Li Jianzhong, Rotem D, Srivastava J. Algorithms for loading parallel grid files. In: Buneman P, Jajodia S eds. Proceedings of ACM SIGMOD'93, Washington DC, 1993, Baltimore: ACM Press, 1993:347—356.
31 Ullman J. Principles of database and knowledge—base systems (Volume 2). Rockvilde Maryland: Computer Science Press, 1989:668—673.

PARALLEL DATA OPERATION ALGORITHMS AND QUERY OPTIMIZATION TECHNIQUES

Li Jianzhong

(*Information Research Institute, Heilongjiang University, Harbin* 150080)

Abstract This paper is the second one of the two series papers that survey the research area of parallel database systems. In the paper, the parallel algorithms for implementing relational operators and the query optimization techniques for parallel database systems are reviewed. To close the two series papers, the future research directions and issues of parallel database systems are discussed at the end of the paper.

Key words Parallel database, parallel algorithms for implementing relational operators, query optimization for parallel database systems.